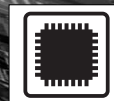
初歩からのHDLテストベンチ

第4回　標準出力の記述方法

安岡貴志

HDLで回路を記述できるようになったばかりで，これからテストベンチを書こうとしている方を対象とした連載の第4回である．前回までは波形での検証結果確認を前提に，基本的なテストベンチの作成方法を解説した．今回は，波形の目視に頼らない検証結果の確認方法として，標準出力の記述方法を解説する．（筆者）

標準出力とは，処理結果を文字列で出力するものと考えてください．UNIXやLinuxなどのターミナル上でシミュレーションを実行しているのあれば，そのターミナル〔**図1(a)**〕です．シミュレーション・ツールで専用ウィンドウを表示しているのであれば，その中でログなどが表示されるウィンドウ〔**図1(b)**〕です．

1．標準出力の書き方

Verilog HDL

Verilog HDLでテキストを出力するためには，システム・タスク`$display`を使います．

図2(a)に`$display`の書式を示します．かっこの中に書かれた信号の値や，ダブル・クォーテーション(" ")の中に書かれた文字列を標準出力に表示します．シミュレーション中にこのタスクが実行されると，その時点の信号の値とダブル・クォーテーションの中の文字列が標準出力に表示されます．

図2(b)に`$display`の記述例と表示例を示します．

ダブル・クォーテーションの中の文字列の中に`%`(とそれに続く1文字)が含まれると，文字列に続く信号名の値が，`%`(とそれに続く1文字)と置き換えられて表示されます〔**図2(c)**〕．

`$display`は，`initial`文や`always`文の中で使います〔**図2(d)**〕．また，ダブル・クォーテーションの中では，特殊文字を使うことができます〔**図2(e)**〕．

VHDL

● **line変数**

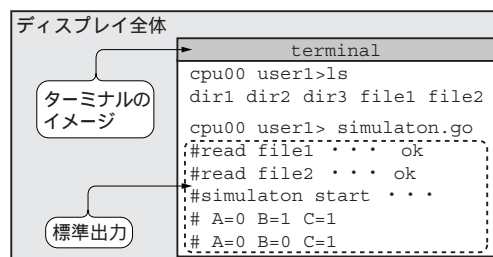
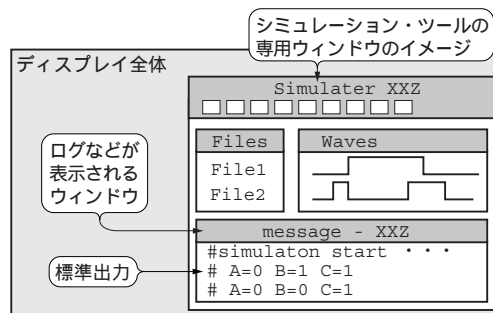
VHDLでテキストを入力したり，出力したりするために

(**a**) UNIXやLinuxのターミナルのイメージ

(**b**) シミュレータ専用ウィンドウのイメージ

図1　標準出力
標準出力は，UNIXやLinuxなどのターミナル上でシミュレーションを実行していれば，そのターミナル，シミュレーション・ツールで専用ウィンドウを表示していれば，その中でログなどが表示されるウィンドウになる．

KeyWord　テストベンチ，テスト入力，絶対時間，相対時間，fork，wait，after，assert文，エンコーダ，for文

```
$display(信号名);
$display(信号名,信号名,・・・);
$display("文字列",信号名);
$display("文字列",信号名,"文字列",信号名,信号名,・・・);
```

(a) 書式

```
module and_comb_tb2();
reg   SA,SB;
wire  SY;

and_comb and_comb(.A(SA), .B(SB), .Y(SY));

initial begin
     SA = 0; SB = 0;
#100 SA = 1; SB = 0;
#100 SA = 0; SB = 1;
#100 SA = 1; SB = 1;
#100 $finish;
end

initial begin
#50  $display("A=%b B=%b Y=%b",SA,SB,SY);
#100 $display("A=%b B=%b Y=%b",SA,SB,SY);
#100 $display("A=%b B=%b Y=%b",SA,SB,SY);
#100 $display("A=%b B=%b Y=%b",SA,SB,SY);
end

endmodule
```

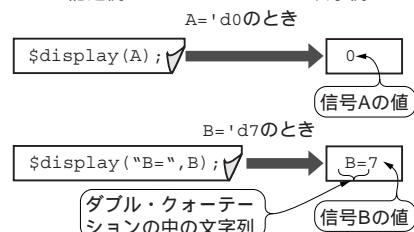
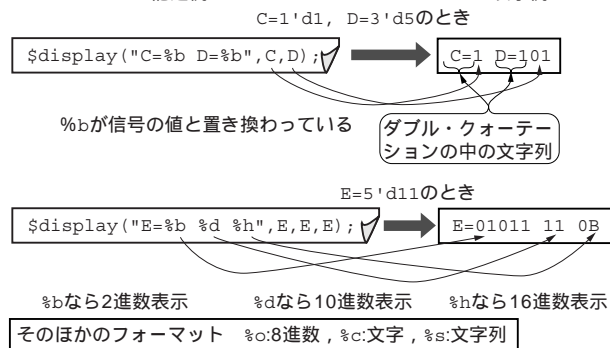
```
module and_comb ( A, B, Y);
input A,B;
output Y;

assign Y = A & B;

endmodule
```

出力結果

```
A=0 B=0 Y=0
A=1 B=0 Y=0
A=0 B=1 Y=0
A=1 B=1 Y=1
```

(d) テストベンチ全体の記述例と表示例

図2 システム・タスク$displayの使い方

Verilog HDLでテキストを入力したり出力したりするためには，システム・タスク$displayを使う．

```
variable 変数名 : line;
```

(a) 書式

```
variable LO: line;
```

(b) 記述例

```
library STD;
use STD.TEXTIO.all;
```

(c) TEXTIOパッケージの呼び出し

図3 line変数

1行分のテキスト・データを蓄えるための変数である．

は，まず1行ごとのテキスト・データを蓄えます．1行分のテキスト・データを蓄えるための変数をline変数といいます．

図3(a)と**図3(b)**にline変数の宣言の書式と記述例を示します．また，line変数を使うには，STDライブラリのTEXTIOパッケージが必要です[**図3(c)**]．

●プロシージャwrite

writeはline変数に値を代入するプロシージャ(関数，もしくはサブプログラムのようなもの)です．writeはVHDLの標準仕様に組み込まれています．

またwriteでstd_logicやstd_logic_vectorを扱う場合には，std_logic_textioパッケージを呼び出す必要があります(**図4**)．

●プロシージャwriteline

writelineはline変数に格納された値を，標準出力，もしくはファイルに出力するプロシージャです．**図5**に

図4　プロシージャwrite
line変数に値を代入する．

図5　プロシージャwriteline
line変数に格納された値を標準出力に出力する．

図6　VHDLによる標準出力のための記述例と動作
プロシージャwriteとwritelineはprocess文の中に書くことができる．

writelineを使って標準出力に出力するための書式と記述例を示します．各プロシージャはprocess文の中に書くことができます(**図6**)．

● string型変数

図6の記述例には，stringという変数があります．プロシージャwriteの引き数には，直接文字列を与えられません．そこで，文字列を出力したい場合には，文字列(string)型の変数を宣言し，これに文字列を代入して，この変数をプロシージャwriteの引き数にします(**図7**)．

図7
string型変数
文字列(string)型の変数を宣言し，これに文字列を代入して，この変数をプロシージャwriteの引き数にする．

2．テストベンチへの適用

あなたが作るテストベンチの検証対象の回路は，ほとんどが順序回路(フリップフロップなどの記憶素子を含む回路)となります[注1]．順序回路の出力は，クロック信号に同期して変化する(主にクロックの立ち上がりで変化する)ので，出力信号が期待通りかどうかは，1サイクル(クロック信号1周期)に1回確認すれば十分です．

● どこでデータを取るか

図8(a)において，検証対象の回路は，クロック信号CLKの立ち上がりで動作しています．信号RST_XとENはテスト入力，CNT4は検証対象回路の出力とします．

ここではデータ(信号の値)を取るタイミングは，クロックの立ち上がりから1サイクルの10％程度前にしています．指定されたタイミングでデータを取ると，**図8(b)**のようになります．クロック信号CLKは，常に同じ周期で‘1’と‘0’を繰り返しているだけなので，データは取っていません．

本来RTL(Register Transfer Level)シミュレーションでは遅延がないので，同じ周期の中であればどのタイミングでデータを取っても同じ(クロックの立ち上がりを除く)です．

注1　本連載の第1回(本誌2007年5月号，pp.70-79)でも説明したように，実際の開発では，ある程度まとまった機能ブロック(数千ゲートから数十万ゲート)ごとにテストベンチを作る．従って，検証対象の回路が組み合わせ回路だけということは極めて稀である．なお，本稿ではテストベンチの解説に主眼をおいているので，全体を把握しやすくするために検証対象の回路を極めて小さくしている．

ただし，ゲート・レベル・シミュレーションでは，各ゲートやセルに遅延が付加されるので，値が安定するのはクロックの立ち上がり直前になります（**図9**）．

データをクロックの立ち上がり直前で取るようにしておけば，RTLシミュレーションでもゲート・シミュレーションでも，同じ周期のデータを同じタイミングで取れるので，結果を比較しやすく，同じテストベンチを使い回せます．

（**a**）タイミング・チャート

RST_X	EN	CNT4
0	0	0
1	1	0
1	1	1
1	1	2

（**b**）出力結果

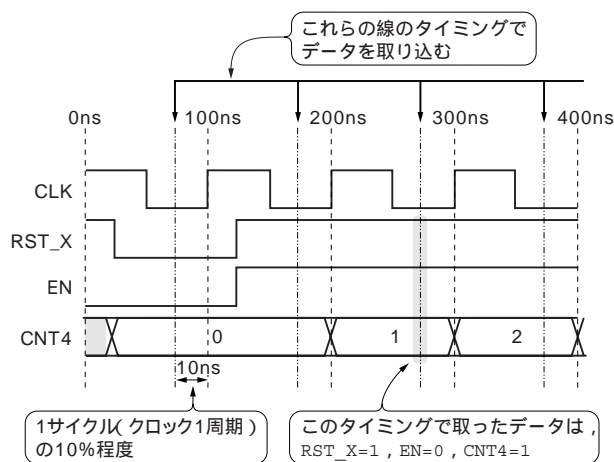



図8　データを取るタイミング（RTLシミュレーション時）
クロックの立ち上がりから1サイクルの10％程度前にする．

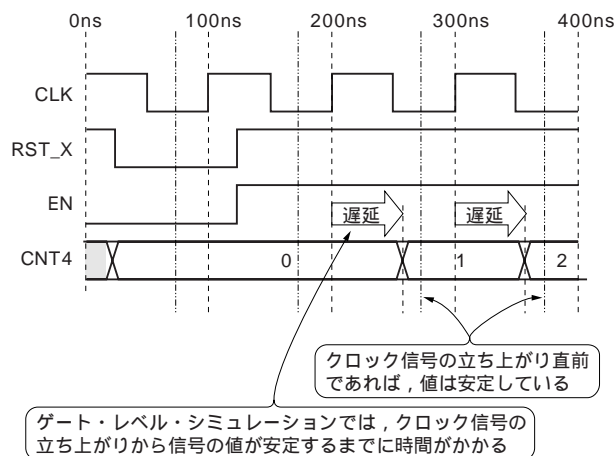



図9　データを取るタイミング（ゲート・レベル・シミュレーション時）
ゲート・レベル・シミュレーションでは，各ゲートやセルに遅延が付加されるので，値が安定するのはクロックの立ち上がり直前になる．

3．標準出力を使ったテストベンチの実際

● 4ビット・カウンタのテストベンチに標準出力を追加

Verilog HDL

リスト1は，連載第2回（本誌2007年8月号，pp.116-124）で作った4ビット・カウンタのテストベンチに，標準出力のための記述を加えたものです（p.129の**コラム**「`timescale」を参照）．

テストベンチの中には，テスト入力のためのinitial文と，標準出力のためのinitial文があります．これらは並行して実行されます．

後者では，パラメータSTROBEでデータを取るタイミングを調整した後（最初の$displayが，クロックの立ち上

リスト1　標準出力を用いた4ビット・カウンタのテストベンチ（Verilog HDL）

```
`timescale 1 ns / 100 ps
module counter_tb;

parameter CYCLE      = 100;
parameter HALF_CYCLE = 50;
parameter DELAY      = 10;
parameter STROBE     = 90;

reg        RST_X,CLK,COUNTON;
wire [3:0] CNT4;
integer    I;

counter counter(.CLK(CLK), .RST_X(RST_X),
                .COUNTON(COUNTON), .CNT4(CNT4));

always begin
              CLK = 1'b1;
  #HALF_CYCLE CLK = 1'b0;
  #HALF_CYCLE;
end

initial begin
  RST_X =1'b1; COUNTON = 1'b0;
  #DELAY;
  #CYCLE       RST_X   = 1'b0;
  #CYCLE       RST_X   = 1'b1;
  #CYCLE       COUNTON = 1'b1;
  #(15*CYCLE)  RST_X   = 1'b0;
  #CYCLE       RST_X   = 1'b1;
  #(5*CYCLE)   COUNTON = 1'b0;
  #CYCLE       COUNTON = 1'b1;
  #(6*CYCLE)   COUNTON = 1'b0;
  #(2*CYCLE)   RST_X   = 1'b0;
  #CYCLE       RST_X   = 1'b1;
  #CYCLE       COUNTON = 1'b1;
  #(5*CYCLE)   $finish;
end

initial begin
  #STROBE;
  for(I=0;I<40;I=I+1)begin
    $display("RST_X=%b",RST_X,
      " COUNTON=%b",COUNTON," CNT4=%h",CNT4);
    #CYCLE;
  end
end

endmodule
```

コラム参照
データを取るタイミングを
パラメータ化
並行して
実行
標準出力のための記述
タイミング調整
$displayの実行と
1サイクルの遅延を
40回繰り返す

がりから1/10サイクル程度前に，呼び出されるように調整した後），for文を使って1サイクル（クロック信号1周期）に1回，$displayを実行しています．

VHDL

リスト2は，連載第2回で作った4ビット・カウンタのテストベンチに標準出力のための記述を加えたものです．

テストベンチの中にはクロックの生成，テスト入力の生成と，標準出力の三つのprocess文があり，これらは並行して実行されます．

標準出力のprocess文の中では，定数STROBEでデータを取るタイミングを調整した後（最初のライン変数への値の格納が，クロックの立ち上がりから1/10サイクル程度前に，実行されるように調整した後），for文を使って1サイクル（クロック信号1周期）に1回，標準出力への出力を実行しています．

● 標準出力による結果の確認

リスト1や**リスト2**のテストベンチでシミュレーションを実行すると，テストベンチ，検証対象回路とも正しく記述できている場合には，標準出力に**図10**のような文字が出力されます．これを参考にすれば，波形を見なくてもバグ解析ができるようになります．

なお，正しい値が出力されなかった場合，テストベンチか検証対象回路にバグがあることになります．この場合，**図11**のようにバグ解析を進めます．

まず，テストベンチからバグがないか確認を始めます．本来の検証は，テストベンチのバグが取り去られ，疑うべきは回路だけにしてから始めます．テストベンチにバグがあるうちは，回路の検証はできないので，最初にテストベンチを完ぺきにしておく必要があります．

図12に，バグ解析における最悪のケースを示します

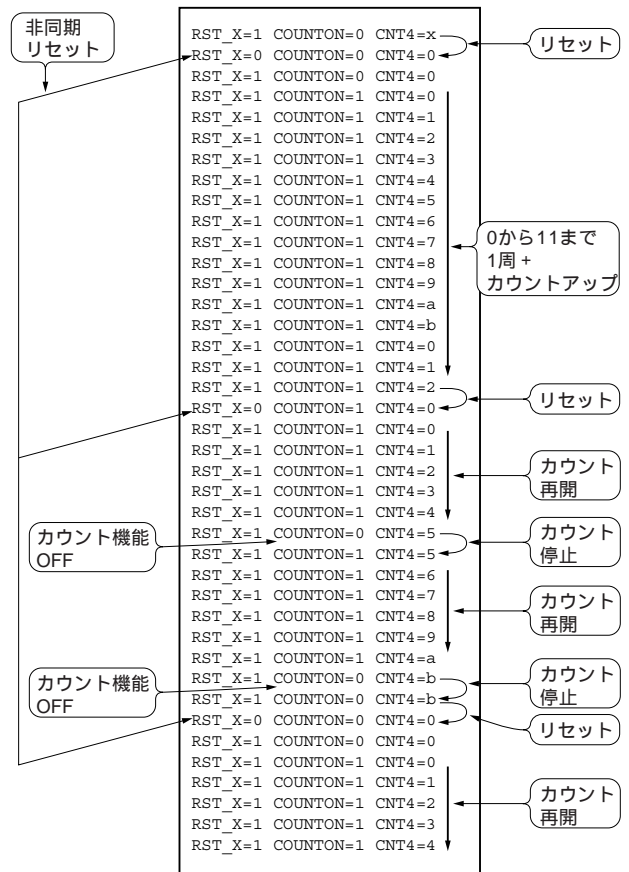



図10　標準出力による結果の確認
リスト1やリスト2のテストベンチでシミュレーションを実行すると，標準出力にこのような文字が出力される．

コラム　`timescale

本文で示した**リスト1**の先頭にある`timescaleは，シミュレーション時刻の単位付けをしています．`timescaleの書式を**図A**に示します．

実はRTLシミュレーションでは遅延を付加しないので，この記述は必要ありません．必要となるのは，ゲート・レベル・シミュレーションからということになります．

なお，この記述は最初に読み込むファイルの先頭（普通はテストベンチ）にのみ（シミュレーションで使用するファイル全体で1カ所のみ）に記述します．この記述は一度設定すると，読み込むすべてのファイルに有効になります．

図A　`timescaleの書式と記述例

リスト2　標準出力を用いた4ビット・カウンタのテストベンチ(VHDL)

```
library IEEE,STD;
use IEEE.std_logic_1164.all;
use STD.TEXTIO.all;
use IEEE.std_logic_textio.all;

entity counter_tb is
end counter_tb;

architecture SIM of counter_tb is

component counter
   port (CLK,RST_X,COUNTON : in  std_logic;
         CNT4              : out std_logic_vector(3
                                          downto 0));
end component;

  constant CYCLE          : Time := 100 ns;
  constant HALF_CYCLE     : Time :=  50 ns;
  constant DELAY          : Time :=  10 ns;
  constant STROBE         : Time :=  90 ns;

  signal CLK,RST_X,COUNTON : std_logic;
  signal CNT4              : std_logic_vector(3
                                          downto 0);

begin

ucounter : counter port map (
  CLK     => CLK    ,
  RST_X   => RST_X  ,
  COUNTON => COUNTON,
  CNT4    => CNT4   );

process begin
  CLK <= '1'; wait for HALF_CYCLE;
  CLK <= '0'; wait for HALF_CYCLE;
end process;

process begin
  RST_X <='1'; COUNTON <= '0';
  wait for DELAY;
  wait for CYCLE;      RST_X   <= '0';
  wait for CYCLE;      RST_X   <= '1';
  wait for CYCLE;      COUNTON <= '1';
  wait for (15*CYCLE); RST_X   <= '0';
  wait for CYCLE;      RST_X   <= '1';
  wait for (5*CYCLE);  COUNTON <= '0';
  wait for CYCLE;      COUNTON <= '1';
  wait for (6*CYCLE);  COUNTON <= '0';
  wait for (2*CYCLE);  RST_X   <= '0';
  wait for CYCLE;      RST_X   <= '1';
  wait for CYCLE;      COUNTON <= '1';
  wait for (5*CYCLE);  assert false;
end process;

process
  variable LO : line;
  variable S6 : string(1 to 6);
  variable S9 : string(1 to 9);
begin
  wait for STROBE;
  for I in 0 to 39 loop
    S6 := "RST_X="; write(LO,S6); write(LO,RST_X);
    S9 := " COUNTON="; write(LO,S9);
    write(LO,COUNTON);
    S6 := " CNT4="; write(LO,S6); hwrite(LO,CNT4);
    writeline(output,LO);
    wait for CYCLE;
  end loop;
  wait;
end process;

end SIM;

configuration cfg_counter_tb of counter_tb is
  for SIM
  end for;
end cfg_counter_tb;
```


(p.131の**コラム**「観察方法によるバグの例」も合わせて参照)．このようにならないよう気をつけましょう．

● 標準出力のための文法

これまで，基本的な標準出力のための文法を解説してきました．この項ではここまで登場しなかった文法をまとめて解説します．

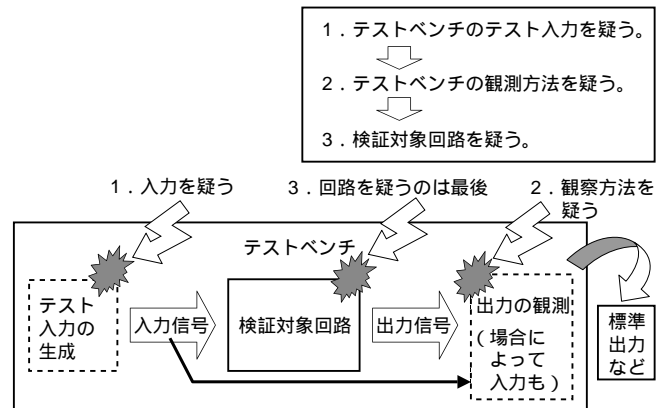



図11　バグ解析の手順
最初にテストベンチを完ぺきにしておく必要がある．

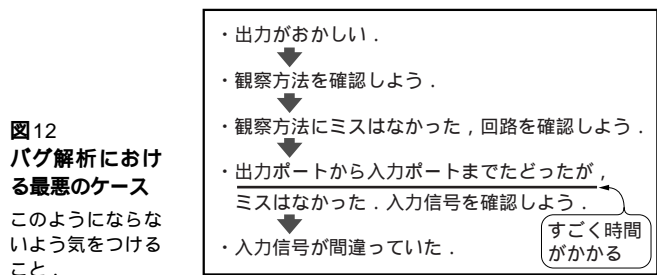
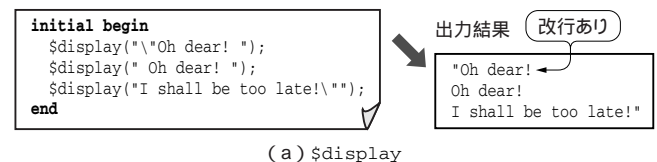
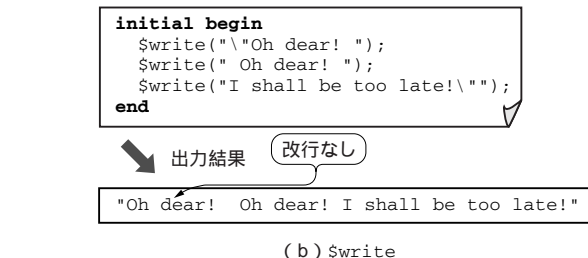



図12　バグ解析における最悪のケース
このようにならないよう気をつけること．

・出力がおかしい．
・観察方法を確認しよう．
・観察方法にミスはなかった，回路を確認しよう．
・出力ポートから入力ポートまでたどったが，ミスはなかった．入力信号を確認しよう．
・入力信号が間違っていた．

すごく時間がかかる

(a) $display

(b) $write

図13　$displayと$writeの出力の違い
$writeの引き数は$displayとまったく同じだが，出力結果の行末に改行がない．

Verilog HDL

● システム・タスク$write

システム・タスク$writeの引き数の形式は，$displayとまったく同じです．$displayとの差は，行末に改行がないことです．

図13に，$displayと$writeの出力の差を示します．

● システム・タスク$strobe

システム・タスク$strobeも引き数の形式は，$displayとまったく同じです．$displayとの差は非常に分かりにくいのですが，$displayが呼び出されたそのときの信号の値を出力するのに対して，$strobeは同じ時間で発生するすべての代入が終わってから出力します．

図14に，$displayと$strobeの出力の差を示します．

● システム・タスク$monitor

システム・タスク$monitorも引き数の形式は，$displayとまったく同じです．ただし，$monitorは一度呼び

コラム　観察方法におけるバグの例

● 1サイクルの遅延の書き忘れ

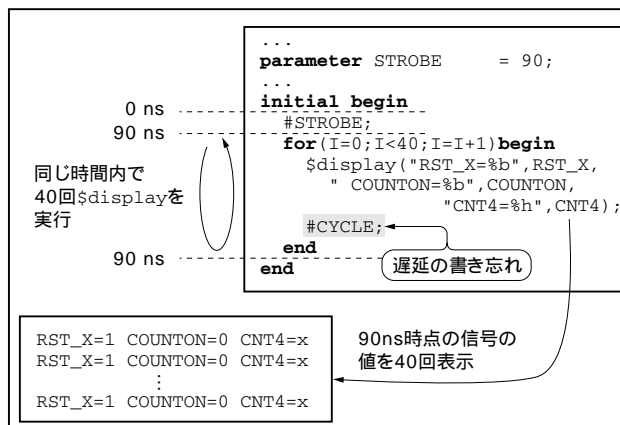
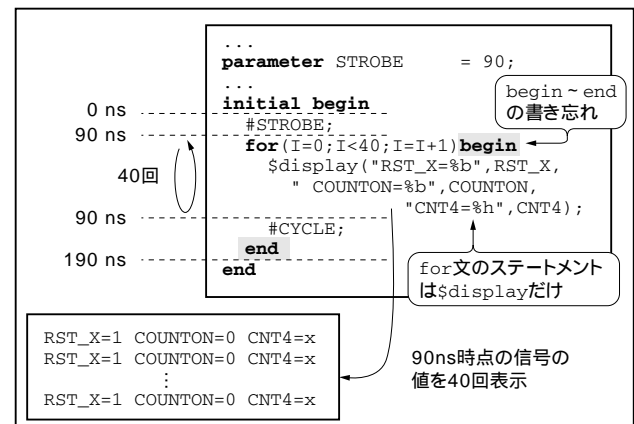
Verilog HDL

リストB-1は，1サイクルごとに信号の値を取るための遅延を書き忘れてしまった例です．この場合，シミュレーション時間90ns時点で，$displayの文を40回呼び出します．シミュレーション自体は4010nsまで進みますが，標準出力には90ns時点の信号の値しか表示されません．

リストB-2はfor文のステートメントをbegin～endで囲うのを忘れた例です．結果は**リストB-1**と同じです．

VHDL

リストB-3は，1サイクルごとに信号の値を取るための遅延を書き忘れてしまった例です．この場合，シミュレーション時間90ns時点で，ライン変数への信号の値の格納と標準出力への出力を，40回行います．シミュレーション自体は4010nsまで進みますが，標準出力には90ns時点の信号の値しか表示されません．

リストB-1　1サイクルごとに信号の値を取るための遅延を書き忘れてしまった場合の例（Verilog HDL）

リストB-2　ステートメントをbegin～endで囲うのを忘れた例（Verilog HDL）

リストB-3　1サイクルごとに信号の値を取るための遅延を書き忘れてしまった例（VHDL）

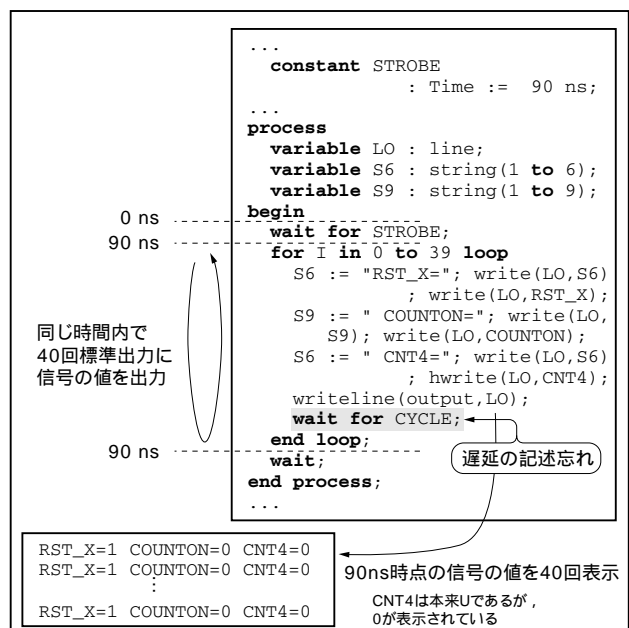
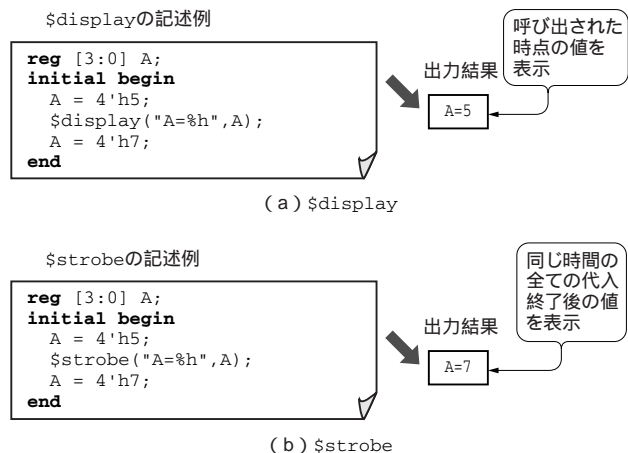



図14　$displayと$monitorの出力の違い
$strobeは同じ時間で発生するすべての代入が終わってから出力する．

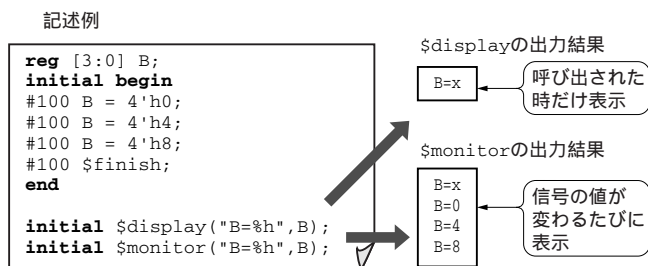



図15　$displayと$strobeの出力の違い
$monitorは一度呼び出されると，指定された信号が変化するたびに表示を行う．

出されると，指定された信号が変化するたびに表示を行います．

図15に，$displayと$monitorの出力の差を示します．

VHDL

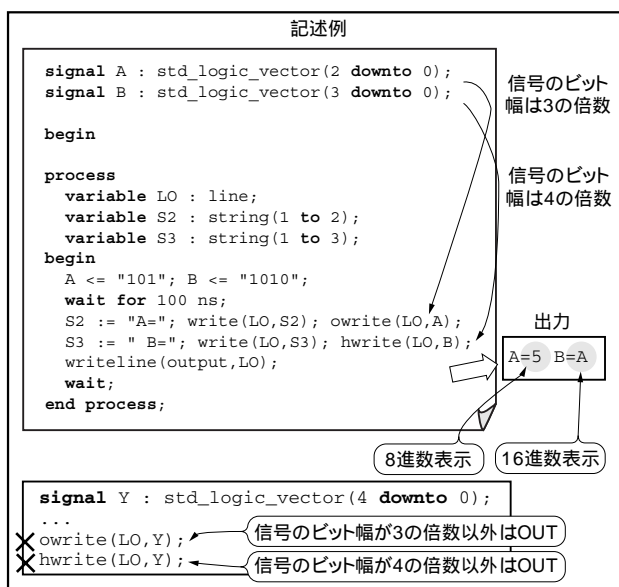
● プロシージャhwrite

プロシージャhwriteの引き数の形式は，writeとまったく同じです．ただし，表示が16進数になります．また，引き数となる信号のビット幅は4の倍数でなければいけません．

● プロシージャowrite

プロシージャowriteの引き数の形式は，writeとまったく同じです．ただし，表示が8進数になります．また，引き数となる信号のビット幅は3の倍数でなければいけません．

リスト3に，hwriteとowriteの記述例と出力を示します．

リスト3　hwriteとowriteの記述例

● まとめ

今回は波形以外のシミュレーション結果の確認方法として，標準出力による確認方法とその文法を紹介しました．ただし，どちらの方法もシミュレーション実行直後に，内容を確認しなければ，結果は消えてしまいます．

実設計においては，回路の完成までに何度もシミュレーションを行い，その結果をテキスト・ファイルに保存しておくという手法をとるのが普通です．結果をテキスト・ファイルに保存するための文法は，標準出力のための文法と非常に似通っています．今回の標準出力のための文法がしっかり理解できれば，ファイルで残す手法も簡単に利用できます．

次回は標準出力のための文法を元に，ファイル出力のための文法を中心に紹介します

やすおか・たかし
(株)エッチ・ディー・ラボ

<筆者プロフィール>
安岡貴志．東京理科大学 理工学部 数学科卒業．前職のデザインセンターでは，3年間Verilog HDLによるASIC開発に携わる．2002年にエッチ・ディー・ラボに入社し，Verilog HDL，VHDL，SystemCによる開発に従事するほか，同社のトレーニング講師を務める．